DOCUMENT RESUME

ED 284 445 FL 016 851

AUTHOR Pusack, James P.
TITLE Problems and Prospects in Foreign Language Computing.
PUB DATE 82
NOTE 28p.; In: Smith, William Flint, Ed. Modern Media in Foreign Language Education: Theory and Implementation. The ACTFL Foreign Language Education Series; see FL 016 850.
PUB TYPE Information Analyses (070)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS *Authoring Aids (Programing); *Computer Assisted Instruction; *Computer Software; Computer Uses in Education; Educational Technology; Equipment Utilization; Individualized Instruction; *Instructional Development; Item Banks; Language Proficiency; Language Skills; *Language Teachers; Pattern Drills (Language); Programing Languages; *Second Language Instruction

ABSTRACT

The problems and prospects of the field of foreign language computing are profiled through a survey of typical implementation, development, and research projects that language teachers may undertake. Basic concepts in instructional design, hardware, and software are first clarified. Implementation projects involving courseware evaluation, textbook packages, generic packages, and orchestrated miscellany are discussed. Development projects described involve drills and games using simple templates, customization of modifiable packages, practice or diagnosis using an authoring tool, short-answer drill using a programming language, self-contained tutorial units, utility programs to manage instruction, branching tutorials using an authoring language, generative drill for simple structural features, simple reading comprehension exercises, and adventure games using preprogrammed models. Among research projects described are those involving comprehensive tutorials with recordkeeping, drill or tutorial with animated graphics, authoring tools for grammar or reading, large databases for structure and reading practice, complex simulations of target culture situations, language understanding projects, and validated placement and proficiency tests. (CB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED284445

Problems and Prospects in Foreign Language Computing

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

James P. Pusack
The University of Iowa

Introduction

High expectations and low expertise mark foreign language teaching in the matter of computing. The profession has progressed since the days when nothing was known and nothing expected, but the gap between available software and the myriad obvious ways computers can assist language learners causes concern. What conditions will lead language teachers to exploit the full potential of technology? How will interested colleagues acquire the knowledge they need to become successful authors of teaching and learning programs? Who will fund such efforts? The purpose of this chapter is to demonstrate that these questions and others like them are worth asking and answering. The problems and prospects of the field are profiled here through a survey of typical projects that language teachers might undertake. The three main sections of this chapter sketch a series of such projects—implementation, development, and research—in order of increasing complexity. A common technological basis will prove useful as the field's problems and prospects are reviewed. This introduction therefore attempts to clarify some basic concepts in instructional design, hardware, and software.

The environment for exploiting computer technology includes more than equipment and knowledge. Current attitudes play a key role. As faculty members become familiar with microcomputers through daily writing at a word processor, basic reservations give way to curiosity. This

James P. Pusack (Ph.D., Indiana University) is Associate Professor of German and Director of the Project for International Communication Studies at the University of Iowa. He is a member of ACTFL, CALICO, and AATG, and co-author of *Using Computers in Teaching Foreign Languages* (Harcourt Brace Jovanovich/Center for Applied Linguistics). DASHER, his foreign language CAI authoring system, is published by CONDUIT, where he also serves as language series editor.

FL016851

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY
L. Fiddle
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

is a positive shift. Administrators, who must provide equipment and released time for major projects, also recognize the potential of technology; the greatest danger on this front comes from those who would promote technology at the expense of the faculty's other priorities. Publishers of foreign language textbooks have embarked cautiously on development projects, but are inevitably guided by short-term market considerations; they view foreign language software—courseware—as an adjunct to their primary product, the textbook.

The area receiving least attention today is the systematic development of the requisite support infrastructure—the experts and facilities that will help faculty create and deliver computer-assisted language learning (CALL) to students. Short-sighted administrators think that a teacher and a machine will work miracles. Far-seeing administrators realize that undertakings like those described below cannot succeed without a staff of language laboratory and computer-center professionals who can translate faculty initiatives into well-designed, carefully implemented software.

The list of objectives sketched here is certainly not all-inclusive. Its hallmark is hard-nosed practicality. These are projects that can be accomplished with today's technology and that can benefit students in systematic rather than haphazard ways. Many other clever tasks can be imagined that put language learners and computers together. The ones chosen for inclusion here strive to improve the productivity of both teacher and learner and thus improve the quality of language learning.

Typologies and Terms

The least edifying environs of foreign language computing lie in the jungle of acronyms. CAI, CBE, CMI, CAFLI, and CALL often fail as signposts because there is little agreement about their meaning; for most listeners and readers, these acronyms differ but do not distinguish clearly among concepts. One group of terms (CAI, CMI, CBE) applies to all forms of instructional computing, not only to work in foreign languages. Computer-assisted instruction (CAI) usually refers to interactive teaching programs, while those programs whose approach derives from programmed learning concepts are sometimes called "traditional CAI." Learner control has begun to predominate in interactive software; as this and other changes take place, CAI as a generic term will probably survive by changing its connotations.

Computer-managed instruction (CMI), in contrast, is used to designate the use of computers to guide students through a course of study, much of which may be "off-line," that is, away from the computer; CMI does extensive recordkeeping to track student progress through traditional or

computer-based materials. Computer-based education (CBE) encompasses nearly everything that students can do to learn via computers: study programming; practice the application of concepts; use word processors to write essays; design graphics; gather and manipulate data with spreadsheets, databases, or statistical programs. CBE (and its alias, CBI, for computer-based instruction) is therefore the most inclusive term.

Foreign language teachers engaged in CAI have long sought a compact way to label their mission. Acronyms containing the letter "I" appear to stress the machine as *instructor;* those with an "L" strive for a *learner*-centered interaction. Thus one can find both CALI (computer-assisted language instruction) and CALL (computer-assisted language learning). CAFLI (computer-assisted foreign language instruction) has also been proposed. Fashion will no doubt lead us in the near future to CALA (computer-assisted language acquisition). In the meantime, the profession's focus on the learner and the benefits of easy pronunciation suggest the use of CALL, pronounced as a word, rather than separate letters. Most of our colleagues, however, cannot be expected to recognize anything briefer than "foreign language CAI."

Workers in the area of CALL have proposed a number of typologies or classification systems that help make some basic distinctions in discussing foreign language software. Many general treatments of computers in education distinguish four fundamental categories: drills, tutorials, simulations, and management (CMI). Pusack and Otto (13) have constructed a matrix of possible applications by examining traditional areas of language teaching (grammar, listening, culture, etc.) in terms of four approaches to CALL: (1) practice and diagnosis; (2) tutorial; (3) simulation and problem-solving; and (4) utility. Each of these approaches is described briefly below.

Practice. Current practice programs usually offer discrete-item drill on a sequence of structural features of the target language. Simple drill programs that present items in a fixed sequence and undertake little processing of student input tend to fall into the oft-scorned class of software dubbed "computer flashcards" or "electronic workbooks." Reflecting the best insights into language-teaching methodology, many CALL projects today attempt to embed all practice in a meaningful context and to treat student responses as linguistically analyzable utterances. Nevertheless, most of the available CALL courseware occupies the very limited space defined by the intersection between the *grammar/structure* area and the *practice and diagnosis* approach. An appropriately constructed bank of practice materials can also be employed as a *diagnostic tool* by the teacher or by learners themselves.

Tutorial. Tutorial programs have the goal of presenting language skills and concepts in a more efficient or effective fashion than textbooks or classroom explanations. These programs exploit the computer's power to

present information dynamically, adjust materials to the learner's own abilities, and keep careful records of student progress. The potential of language tutorials has not yet been fully realized and to date there are no complete, commercially available CALL tutorials that could be used, for example for home study of the basics of a foreign language.

Simulations and Problem-solving. Simulations place learners into a small model of reality in order to expose them to cultural content or foster use of the target language in a lifelike context. In most simulations, the task posed by the simulation takes precedence over correct production of the target language; the emphasis is on comprehension and meaningful expression. Adventure games fall into the category of simulation. Problem-solving programs likewise pose a task, but may not stress the student's involvement in a fictional situation as strongly as do simulations. Reading and listening comprehension programs that require the learner to extract complex information, often involving critical thinking skills, can be viewed as problem-solving software. The boundary between simulations and problem-solving depends on the level of illusion the learner is willing to accept and the degree of control the language learner has over his or her progress through the tasks offered by the computer.

Utility. Utility programs place a very high priority on the ability of students to make decisions in using the computer for language learning. Word processing (e.g., target-language composition) is the most familiar example. The emerging generation of software will bring with it many programs that give students tools for evaluating their own writing, for reading texts more efficiently, and for diagnosing their own weaknesses. Even tutorials, which are classically the most directive and prescriptive kind of software, will be designed explicitly as complex learning utilities that the learner can control fully and therefore ignore selectively.

It should be clear by now that the above four terms constitute not only categories, but also characteristics. For example, a given piece of courseware may have elements of both the tutorial and the problem-solving approach. Authors of software increasingly blend these various design characteristics into their products in order to increase their effectiveness and appeal.

Developing foreign language courseware demands expertise. Teachers in the field generally recognize that high-quality software will not spring from their own efforts to learn computer programming. In describing any development model, it has become accepted practice to refer to three discernible roles: the *subject-matter specialist* (or content expert), the *instructional designer,* and the *programmer.* In addition, graphics design expertise has become increasingly important. For CALL, this tripartite division of labor suggests that the language teacher will provide linguistic and pedagogical expertise, while the programmer will cause the computer to carry out the desired tasks. The instructional designer understands

both programming and teaching and can therefore improve the working relationship between the other two. In addition, the instructional designer brings to a CAI project a storehouse of experience with instructional software; this critical knowledge forms a starting point for any project that aspires to break new ground and meet current expectations. The following sections assume no particular developmental model; rather, the roles of the instructor/author, the instructional designer, and the programmer at times coalesce. In some cases, however, distinctions are made that derive from the complexity of the task under discussion.

Hardware

Hardware intimidates everyone. No matter how much one learns about products, technologies, vendors, prices, features, and the like, it remains too much and not enough. There is too much information to evaluate in terms of pedagogy and never enough to guarantee that one is up to date. Industrial laboratories already hold technologies that will not be advertised for three years or used by the public for five. What level of information is then appropriate?

One must be guided by one's institution and one's peers in the use of *mainframes, minicomputers,* and *microcomputers.* Fortunately, each school and college supports a small set of machines. There may be a large mainframe computer or a powerful minicomputer with many terminals attached; there may be a small microcomputer cluster used by students from many departments. The national standard suggests a mixture of both, with the microcomputer sometimes acting as a timesharing terminal in order to help in the transfer of programs and information. If one's institution does not appear to be at the cutting edge, especially in technologies that apply to CALL, national conferences offer the best opportunity to see vendors' wares and the burgeoning number of colleagues' projects.

Inevitably, language instructors are concerned with various *supporting devices* used to print text, store large amounts of information, and foster communication among teachers and students. The hardware needed for these tasks becomes more economical and powerful with every passing month. For example, the first Apple® computers came equipped with 16 kilobytes (K) of memory—only a few bold purchasers bought additional memory chips; today, it costs less than $50 to add 400K to the memory of an IBM PC-compatible microcomputer. Paradoxically, the longer one waits to acquire a given device, the more likely it is that a better alternative will appear on the market. Delay can be beneficial as well as frustrating.

Printing foreign language characters no longer poses a problem for modern computers. Teachers who use a word processor will find that the publisher of good software will tout its foreign character capabilities and will therefore be able to indicate which printers can be used. Vendors, of

course, should be able to demonstrate the desired capabilities. Instructional software—in contrast to software for research and teaching support—seldom requires a printer. A student workstation should be equipped with a printer only if students will be engaging in composition on the computer, in which case a version printed on paper is usually required; even in this case, fewer printers than computers will be needed.

Most microcomputer systems store programs and data on diskettes, which are useful for personal transfer of information, but have limitations. They are easily damaged, pirated, or stolen; they may not have enough capacity for some programs; and they require that someone issue and collect them. The emerging alternative to diskettes is a larger storage device, the magnetic hard disk or CD-ROM optical disk drive, which may be linked by wires to a whole group of microcomputers. A network recovers on a small scale the advantages of minicomputers and mainframes. Students may have their own data diskettes, but obtain access to programs and large information bases via the network. Networking also allows students to exchange information with each other at their individual microcomputer workstations. This option stimulates the development of software for meaningful student-to-student communication. Unfortunately, maintaining a microcomputer network requires a much higher order of expertise than that needed for a simple, self-contained machine.

Traditional keyboards, augmented by various arrows and function keys, place limits on the kinds of interaction that can be designed. Reading practice takes on an entirely different aspect if learners can express their understanding by indicating (via a pointing device) which portions of a text meet some criterion, as well as simply typing characters. Simulations and games work much more effectively if students can relate their responses directly to positions on the two-dimensional screen display, rather than indirectly via the keyboard.

The most familiar pointing device is the *mouse,* which relates motion of a tracking device on a table to motion of the cursor on the screen. An "inverted" mouse, called a *track ball,* sets up a similar analogue between hand movement and screen; *joysticks* do the same; there are even infrared devices that base their effects on head movement. Two very different technologies, the *light pen* and the *touch screen,* allow the learner to engage in fairly intuitive hand/arm motions relative to the screen. Although not yet in widespread use for CALL, pointing devices are likely to grow in popularity, especially as they become more widely used for word processing. This trend should have a positive impact by allowing graphics to convey CALL tasks in a more faithfully represented context and by enabling language learners to interact more intuitively with texts that develop comprehension skills.

Software

There is really only one issue that defines the current status of foreign language computing: the quality of software. While hardware dictates to some extent what software can do, the language-teaching profession is so far behind in realizing the capabilities of current inexpensive technology that hardware limitations must be considered moot. Of course, new applications with speech synthesis, speech recognition, or even holographic projection can be imagined, but each impulsive leap forward to a new technology will cause one to ignore valuable opportunities with the machines now on student desks. What follows is an overview of the major software options available to language teachers using basic computer equipment. (See also Chapter 4 by Wyatt in this volume for another perspective of CALL software; a second volume of *Modern Media in Foreign Language Education* will hold chapters that deal with development of courseware for audio and video devices.)

A commercially available, documented, and supported piece of software is called a *package*. It may contain one or more programs, consist of several modules, and come on more than one diskette, and it may or may not be copiable. From an instructional point of view, a package can be viewed best as the realization of someone else's language pedagogy. Since language teachers subscribe to a wide variety of methodologies and are often in disagreement about which is the most efficient or suitable approach for a given audience, it should be clear that no courseware package is likely to be judged satisfactory in all respects. Most available packages, for example, offer drill; teachers who despise drill will find them unacceptable. On the other hand, what might be called second-generation CALL software has started to come of age. The first generation was developed chiefly by individuals working alone; it found its way into the hands of publishers and eventually became available commercially. Second-generation packages may be the intellectual product of individuals, but they have been developed by larger teams working to overcome many of the weaknesses of early packages. This approach usually results in better instructional design, more effective graphics, and more sophisticated handling of student responses.

All software requires *programming*, and original forms of instruction require original programming. Since most language teachers are not programers, programming has become the bugaboo of CALL. The solution is straightforward: language teachers should author software, but they should not program it. It is tempting to let the dictum "Don't program!" stand without further comment. There will certainly be exceptions, but more projects will benefit from assuming this rule than from expecting the opposite. There is much to be said, however, in favor of language teachers gaining a basic understanding of elementary programming techniques in

the context of a broader introduction to CALL. Without the demystification that a taste of programming can bring, the CALL instructor/author may develop false expectations about the nature of computer logic and intelligence. (See Chapter 10 of this volume by Curtin and Shinall for an expanded view of this issue.)

Any potential CALL instructor/author should consider developing software using an *authoring system*. An authoring system is essentially a package empty of lesson content. The instructor creates courseware by supplying certain kinds of linguistic information. In the case of CONDUIT's DASHER (12), for example, the teacher must type exercise instructions, questions, and answers; the program then handles presentation of the items and evaluation of the student response according to a predetermined algorithm. A simulation requires scenes, actions, and objects; a reading program requires texts, glossaries, and comprehension goals; structural practice must be supplied with sentence-generation tasks and solutions. Once the materials have been inserted into the authoring system, a fixed set of interactions is presented to the learner. Under these conditions, the instructor/author has complete control over the content but virtually none over the program logic and operation. More extensive criteria for high-quality foreign language authoring tools for CALL have been developed by Otto and Pusack (9).

The advantage of using an authoring system is that the author can apply his or her language-teaching expertise intensively without concern for problems of programming; this is extremely efficient and can lead to the development of large amounts of high-quality courseware in a short period of time. The disadvantage lies in the constraints imposed by the authoring system. The system must offer a valuable teaching/learning strategy; if it does not, no amount of authorial creativity will compensate for this flaw.

The term *template* (pronounced TEM-plit) is virtually synonymous with authoring system. It is usually applied to highly structured authoring mechanisms, such as vocabulary drill programs, into which one must insert lists of language data. Since some commercial packages allow the user to replace the original content with material supplied by the instructor, these packages can also be viewed as templates.

Authoring languages occupy the middle ground between programming in a general computer language like BASIC or Pascal and authoring systems. They offer a set of high-level features useful in authoring instructional courseware but require that these features be programmed by assembling a correct and logical sequence of instructions. Designers of authoring systems attempt to give an author the built-in power of preprogrammed features combined with the flexibility to design his or her own interaction with the learner. Authoring languages usually offer extremely valuable assistance with difficult programming tasks, including graphic display (including character sets), control of audio and video devices,

evaluation of student responses, branching, and recordkeeping. The most advanced authoring systems also permit the author to use general programming languages when needed. This vital option means that any straitjacketlike effects of the authoring language can be overcome with the assistance of a programmer.

PILOT is the most widely known authoring language for microcomputers. Implementations of PILOT differ greatly, so care must be taken to discover which is the best version for a given computer. Many authoring languages have been designed to support traditional frame-by-frame CAI of the programmed-learning variety. These languages work well for industrial, business, and military training purposes, but may have little to offer the foreign language author. On the other hand, some authoring languages boast features designed to support foreign language instruction; *TenCORE* (15), for example, has excellent graphics for foreign-character display and a built-in answer-evaluation routine derived from the PLATO system. Because there is a business market for CAI authoring languages, teachers may discover that potentially interesting products have prohibitively high prices. Even at prices in the thousand-dollar range, however, institutions should be able to justify purchasing a carefully chosen authoring language on the basis of the time and effort it can save. For this reason, a very experienced software author or programmer will also consider using an authoring language if it meets the needs of a given project.

The basic concepts of design, hardware, and software described above will become more tangible as they are applied to specific examples of courseware useful for CALL. The following section sketches a series of introductory-level projects that can be accomplished by any dedicated language instructor.

Implementation Projects

For many faculty, the first approach to CALL should probably be the implementation of an existing, self-contained piece of courseware. A straightforward "implementation project" allows language teachers to overcome some of the basic hardware and software obstacles. First, teachers will find that it is difficult to gain consistent access to hardware without demonstrably useful courseware; on the other hand, without a minimum of several machines for faculty and student use, it is difficult to explore and integrate software at any level. Second, even if computers are available, their effective use requires that teachers develop expertise in software evaluation. Yet most institutions do not have a wide array of software for teachers to examine as a preliminary step to determining whether they should develop new materials or adopt existing packages. There is no simple solution. A sustained effort to read reviews, attend workshops, visit software booths at conferences, and travel to other institutions can

have a cumulative positive effect in preparing any language teacher to decide on an approach to this technology. As a first initiative, the safest may be an implementation project, that is, one that requires no programming or authoring, but rather involves using an existing system or piece of courseware.

Implementing software takes effort and patience. The more interactive and comprehensive a good instructional package, the more likely it is to contain elements not completely congruent with a given teacher's methodology. It is safe to say that *no* computer program taken over from another source will be completely satisfactory. As a result, passions often run high in the critique of software; seldom does one hear such unequivocal—usually negative—statements about textbooks as one hears about computer software. This can be viewed as healthy skepticism, but ought not to lead teachers who would use CALL to reject all available options and plunge immediately into ambitious new projects. Furthermore, when an acceptable package has been found that is compatible with one's available hardware, the work is still not over. The correlation between one's syllabus and the package must still be worked out in detail. (Ariew discusses the ramifications of this process in Chapter 2 of this volume.)

What kinds of software could form the basis for a worthwhile first use of CALL? Given the state of the art, there is a dearth of materials that can make a significant impact on the curriculum. Catalogs of software for a given foreign language reveal very few packages that form, in themselves, a coherent contribution to students' learning. A few miscellaneous programs provide a change of pace, a motivational supplement, or a bit of work on vocabulary, but these disjointed components cannot be assembled easily into a viable strand in the syllabus.

The issue here is whether the effort that goes into acquiring expertise and marshalling a support system for CALL can be justified unless some major impact on the curriculum can be felt. While machines cannot replace teachers, teachers should be able to sense a qualitative difference over the long term in their students' learning. This means that any project—from basic implementation to full-fledged research—should be based on some larger curricular objective. Such objectives might be increased conversation in class because drill is conducted by machine, an added reading component to an otherwise conversation-oriented syllabus, or more time available for the teacher to prepare classes because homework and individualized tutorials are delivered via computer. Realistically speaking, such encompassing goals cannot be accomplished overnight, but we should keep them in focus as we decide where to invest effort in CALL.

Courseware Evaluation

Since implementation projects rely on available courseware, expertise in judging the quality of a package ranks high among the basic skills needed for teachers to inaugurate CALL. Four major concerns deserve careful consideration in package selection: *content, approach, design,* and *delivery.* Unfortunately, evaluation by a simple checklist technique often fails; many dimensions must work in concert to form an effective piece of courseware. The best way to determine the value of a package is to confront it with one's own objectives, use it with students under real learning conditions, and compare it with the alternatives. Compromises of one sort or another can seldom be avoided. In this sense, evaluation of courseware differs little from textbook evaluation. Language teachers need practice to do both tasks fairly and productively.

Content. The content area addressed by a package is always the starting point in evaluation. Advanced reading activities will scarcely benefit first-year students. Arcade-style vocabulary games will probably alienate most graduate students striving for Ph.D. reading proficiency. The capacity of the package to complement or parallel the textbook should also be considered carefully, although this topic is difficult to treat thoroughly without actually executing every module, lesson, or exercise. An increasingly important final issue confronts every serious user of computer software—authenticity. Video, audio, and textbook materials in language courses strive for more and more authentic use of language, where possible, in a communicative context. Many software packages will be found wanting on this score.

Approach. The approach used by the developers of a package reflects their linguistic, pedagogical, and programming knowledge. No single set of questions, however, applies to every package: utility, problem-solving, simulation, tutorial, and drill programs all approach learning with different strategies that must be fully fathomed before their effectiveness can be judged. The pace and liveliness of the courseware, the degree to which it interacts with the learner, and the clarity of instructions exemplify the kinds of questions one can put to any package.

One specific topic relevant to many CALL packages is the way the program handles natural language. Some kinds of language learning, to be sure, lend themselves to dichotomous choices (yes/no, right/wrong, true/false), or multiple-choice treatment of student input. Many simulations, for example, achieve a level of design complexity that compensates for the use of such techniques; one fine example that can be used in the ESL classroom is the package *Where in the World Is Carmen Sandiego? A Mystery Exploration Game* (2). Nevertheless, language teachers have every right to expect CALL software that attempts to analyze student responses, diagnoses or predicts typical student errors, uses linguistic rules to parse student

input, and ultimately understands what the learner says to the computer. Pusack (11) analyzes the nature of answer processing and error correction in CALL, with special attention to various kinds of diagnostic feedback.

Design. The detailed design of a package may have a great influence on its effectiveness under real learning conditions. Our standards in this area rise as we gain experience from both instructional programs and general-purpose software, such as word processors. The visual appeal of the program, its ability to adapt to various learning situations, the degree to which records are kept on student performance, and the program's ease of use all deserve scrutiny once it has been determined that the content and approach of a package meet one's needs.

Delivery. The final criterion for evaluation, delivery, comprises a whole series of questions that should be asked last, but must usually be asked first. By asking which equipment a given package will run on, one funnels the pool of available software to a disappointing trickle. Other typical delivery issues relate to copy protection, educational discounts, the quality of documentation, and the need for special equipment (pointing devices, extra memory, etc.). The realities of academia dictate, unfortunately, that many teachers must launch CALL by determining just what courseware fits their machine, rather than first evaluating their precise curricular needs.

Hope, Taylor, and Pusack (5) offer a comprehensive checklist of evaluation criteria (pp. 80–85). Cornick (3) has developed a point-scoring system for foreign language software evaluation that covers five major areas in detail: content, support material, presentation, stimulation of student interest, and computer techniques. Hamerstrom, Lipton, and Suter (4) specifically address their comments to the high school teacher who is evaluating software and compress their results into a handy half-page form treating content, features, instruction/documentation, creativity, and technical quality. (See Chapter 9 in this volume, by Hubbard, for further discussion of courseware evaluation.)

Textbook Packages

The first and simplest approach to implementing CALL probably lies in using a commercial software package designed to accompany the course textbook. Over the past two or three years, large textbook publishers have become active in developing or sponsoring various kinds of CALL. Often such packages consist of multidisk sets of exercises tied closely to the text or even drawn from it. When exercises closely follow the text, instructors who want to substitute computer homework for paper-and-pencil homework will find their needs are met. Some publishers' materials are based on the textbook but offer language practice beyond what the text provides;

instructors who seek to broaden students' opportunities to use the language outside of class will be well served by such packages. This category of software often includes vocabulary practice with graphics, tutorials, cultural simulations, and conversationlike activities that allow students to apply their skills in meaningful situations.

Unfortunately, publishers are just beginning to invest in software to accompany their foreign language textbooks. Complicating the issue is the fact that, as the methodology of language teaching comes to stress communicative over manipulative practice, the development of good software becomes more expensive. Publishers offer textbooks with up-to-date methodologies, but there are few models of good software to stimulate conversation, build reading skills, or engage students in motivating cultural situations. Even when good software concepts become clear, publishers will find that the cost of worthwhile courseware beyond the level of drill may be prohibitive. Finally, commercially developed packages that employ interactive video and audio are unlikely to appear at all because the available base of hardware is too restricted to justify the expense: too few schools could afford the workstations. CALL with these interactive media will depend on noncommercial efforts by universities and government agencies.

Generic Packages

At the same time that established textbook publishers are exploring ancillary software packages, newer software companies have begun to offer "generic" packages, which represent a second option for those considering an implementation project. Such packages consist of tutorials or exercises suitable for almost any beginning or intermediate curriculum and therefore represent another kind of implementation project. These packages are usually the product of university-level efforts in foreign language CAI. Some may have their origins in early software developed on mainframe computers as far back as the 1960s. Generic tutorials are obviously attractive to software publishers because the market for generic tutorials is not limited to institutions that adopt a specific foreign language textbook. This fact usually works against generic packages being offered by textbook companies.

Generic packages are touted as "stand-alone" materials usable by any student. In practice, it is virtually impossible to create either tutorials or exercises that can be used equally well with all commercial textbooks or that mesh fully with every instructor's syllabus. The sequence of grammar and vocabulary presentation has not reached canonical status, so learners who attempt to use stand-alone packages will find themselves continually frustrated by unfamiliar words and stymied by the courseware's assumptions about known grammar. The teacher who hopes to implement a

generic tutorial or exercise package must carefully review the material to establish which units of the software correspond to which units of the textbook.

The control of vocabulary in large stand-alone packages is a critical element. Even when the package author has made a concerted effort to hold vocabulary to a very small corpus, students in the early weeks and months of a first-year course will still encounter unfamiliar material. On the other hand, more sophisticated packages typically offer at least one solution to this problem by providing interactive on-screen glossaries for students to call up whenever they encounter unfamiliar words.

Grammar sequencing does not lend itself easily to on-screen help. If stand-alone tutorials and exercises are composed of relatively short, self-contained units and offer learners easy mechanisms for selecting appropriate topics, students may be able to tolerate some discrepancies between the software package and their textbook. Whenever computer work is constructed out of explanations or operations that rely on knowledge of one linguistic feature—a tense or a case—to teach another, stand-alone packages will be unacceptably frustrating for elementary students. On the other hand, intermediate students engaged primarily in review and reinforcement of previously studied concepts should not be susceptible to these incongruities between syllabus and software. This may mean that implementation of a stand-alone package is best exercised in the intermediate curriculum.

Orchestrated Miscellany

The third option for initial implementation of CALL consists of the arduous and treacherous journey through the forest of software miscellany. Careful review and screening of small packages from a variety of software houses may yield enough usable material to form a regular—if not completely coherent—computer component for an elementary or intermediate course. The advantage of this approach is that the limitations of one package may be overcome by the strengths of another. One week, students may engage in small-group work with an adventure game as the basis for conversation or composition; the following week, reading comprehension activities may inform students about peculiarities of behavior in the target culture; and the third week, a difficult grammar topic may be treated through individualized tutorial and practice at the computer. Testing and placement may be aided by use of a computer-based achievement test. Composition can be stimulated by introducing students to the use of a foreign language word processor; in this area, newer and better options come onto the market almost monthly.

An eclectic approach also permits growth from semester to semester as new packages become available and as packages acquired previously lose

their appeal after students use them. Those who implement large-scale textbook or stand-alone packages seldom have this freedom to reject unacceptable segments; they may feel locked into a decision made when their expertise was low because no new software has appeared that offers the same organizational advantages as their current courseware without the pedagogical drawbacks.

The above series of implementation projects illustrates the fact that valuable and productive work in CALL need not become a language instructor's all-encompassing concern. The next step beyond imaginative use of others' courseware, however, demands a commensurate increase in resources of various types.

Development Projects

The second level of work in CALL begins with the actual development of new coursework. Although the border is not always distinct, the characteristics that distinguish the tasks of implementation from more ambitious development projects all relate to the kinds of *effort* and *expertise* required. An implementation project focuses the instructor's effort to adapt the curriculum, the syllabus, even the daily lesson plan; the expertise needed lies well within familiar pedagogical and methodological terrain. Anyone embarking on a development project, however, accepts some degree of involvement in the unfamiliar process of authoring software. This will entail various aspects of instructional design, construction of content material, use of an authoring tool or programming language, selection of hardware, and other related tasks. The exact expertise required will depend on the developmental environment at one's institution.

The following sections illustrate the full range of projects that can be considered developmental. These are described in order of increasing complexity, beginning with small-scale work involving simple changes performed on off-the-shelf software. At the high end of the developmental range come large-scale projects that border on the experimental, but involve few risks or problems. The detailed illustrations both amplify the concept of developmental projects and stimulate development efforts.

Drill and Games Using Simple Templates

A number of commercial packages offer a good first approach to developing software by allowing the instructor/author to fill in simple lexical or grammatical materials into very narrowly defined templates. Most of the teacher's decisions relate to content, rather than program operation. The

actual interaction between learner and computer may well be quite complex, include communicative tasks, or depend on very complex programming and design techniques, but the instructional content is simple enough to permit the teacher to "plug in" new elements based on his or her own syllabus. The result is a tailored, course-specific program or package that closely follows the structural and lexical progression familiar to one's students. Of course, the final product will be only as good as the original template.

Customizing Modifiable Packages

Modifiable packages consist of more comprehensive tutorials or exercise collections that span the entire breadth of a course or textbook. Modifying them to suit a particular syllabus encompasses more than simply supplying lists of words and expressions. In some cases, the content of whole instructional segments may require adjustment. Grammar explanations may need to be brought into conformity with the textbook, and the sequence of units must be carefully examined and reconfigured to reflect the week-by-week progression of topics. The complexity of this task depends on the way the original package was designed and created. If modifications of this sort were foreseen by the authors of the original software, it may be possible to reconfigure significant aspects of the package by responding to various prompts or by using other parameter-setting mechanisms. If the package was created from a well-documented authoring system or language, changing textual elements in the courseware may be relatively easy and serve as an excellent introduction to how instructional programs are designed. Software written in general-purpose programming languages like BASIC, Pascal, and C may be modifiable by experienced programmers if the line-by-line operation of the computer code is extraordinarily clear and contains extensive annotation explaining its operation. This is seldom the case, however. Programmers usually regard tinkering with other people's programs as distasteful and unrewarding.

In general, software that is not expressly advertised and distributed as modifiable will probably not offer proper access for modification: license agreements may explicitly forbid changes, disks may be locked by protection schemes, or the computer code may be stored as low-level machine language that would defeat all reasonable attempts to make even small changes. Noncommercial software acquired from colleagues at other institutions may lend itself to modification if proper program documentation is available.

Creating a Test Item Bank with Existing Software

Large computer systems have long offered teachers the use of sophisticated test-generation software. Such programs support the creation of large pools of test items, the printing of tests according to very detailed item-selection criteria, and posttest recordkeeping on the effectiveness of the tests. As similar software becomes available for microcomputers, the use of item banks will become an attractive option for teachers. The tests can be administered either at the computer or with machine-readable forms and other paper-and-pencil methods. Depending on its design, a comprehensive test bank can be used for student placement, regular in-course achievement tests, and diagnostic purposes when students beyond the elementary level need advice about topics for review and remediation.

The task of creating an item bank using an existing system can be an effective first approach to CALL for several reasons: item banks are expected to grow over many semesters; computer-generated tests can be administered without a large number of machines if paper-and-pencil versions are used; and the time savings for the teacher can be felt very soon in the development process. Item banks have a number of disadvantages as well: if they are not administered on the computer ("interactively"), extra work or equipment may be needed to record student answers on a separate medium in order to analyze the items' effectiveness; most teachers have no special training in designing reliable test items; machine testing often implies the use of multiple-choice formats that allow little room for variation in testing formats, especially those that strive to measure communicative ability.

Practice or Diagnosis Using an Authoring Tool

A comprehensive set of original practice materials in reading or language structure can emerge from work with an authoring system or language. The authoring system will limit the scope of the exercises but allow the instructor/author enough freedom to focus chiefly on the approach and content of the materials. An authoring language requires much more time to master for those involved in creating the materials. Even if the language teacher serves only as subject-matter expert and can direct the services of graphics specialists, instructional designers, and programmers, the capabilities of the available authoring language must be fully understood, an appropriate language must be chosen, and prototype lessons must be written. In some instances the authoring language may not have all the capabilities demanded by the instructor/author. In this case either specialized programming will have to be commissioned, or the ambitions of the author will have to be brought back to reality. While working in a typical

authoring language, teachers will frequently find it impossible or extremely difficult to engage in procedures that exploit the language-generating and language-parsing capabilities of modern computer systems. Nevertheless, powerful and useful sets of lessons can be generated with this approach. The work will doubtless be highly labor-intensive, since each lesson may consist of a set of instructional frames that must be composed one by one on the basis of very explicit designs prepared by the foreign language teacher.

Short-Answer Drill Using a Programming Language

The advantage of creating new materials using a programming language lies in the opportunity to devise entirely new types of interactions. Although work in an authoring language often permits a wide range of formats, programming in BASIC or Pascal or another general-purpose language opens up new kinds of access to the power of the machine to analyze student responses. This approach commits the project to acquiring a new level of expertise, either by having participants learn to program or by hiring a programmer. The results of such a project may be far more humble, at first, than the extensive materials that could flow from work with an authoring system over the same period of time. Original programming should be viewed therefore as a long-term investment that serves as the basis for interesting future endeavors, whether or not these have been fully spelled out.

Self-Contained Tutorial Units

CALL tutorials on aspects of structure or culture can be created in relatively efficient fashion using an authoring language. Since most tutorials contain some level of practice within their sequence of activities, projects of this sort constitute a step beyond practice programs. If the tutorials amount to nothing more than a few screens of information linked by instructions to press the RETURN key, their usefulness will be minimal. Good tutorials rank among the most challenging of projects because each topic must be molded into a stimulating, individualized, appealing, and foolproof whole. Foreign language faculty members who would not consider writing a textbook (or at least a textbook chapter) should probably steer away from tutorial projects, since interactive tutorials demand a large amount of developmental time in relation to the resulting student learning time. Informal estimates of this ratio put authoring time at anywhere from 400 to 500 hours for every hour of lesson time. The more animation and other graphics, remedial help, alternate forms of explanation, and sequenced practice activities that are included in the tutorial

design, the greater must be the author's effort for a single unit. For this reason, language faculty may be advised to embark upon a tutorial project by commencing with a few topics that inevitably baffle most students (difficult tense forms, complex agreement problems, confusing prepositional usage) rather than with Week One of the syllabus. A topic like the present tense of regular verbs will seldom justify extensive authoring time and is also unlikely to serve as a sufficiently complex prototype for further developmental process.

Utility Programs to Manage Instruction

Those who want to build programming skills over the long term should consider writing a small set of programs generally useful in support of instruction. Such projects might well be tools used by the teacher to prepare materials or reduce burdensome paperwork. Examples are quiz generators that select vocabulary at random or print tables of verb forms for students to fill in as a quick check on their mastery of morphology. Although grade-book programs are commercially available, the instructor/programmer may find benefits in writing programs that help perform weighted grade calculations and print out scores and comments to students. In the area of translation or composition, programs that take several student's responses to a given task and assemble them on a single handout for class discussion can serve as simple projects that hone the programming skills useful in all foreign language programming.

Branching Tutorials Using an Authoring Language

Branching tutorials that carefully sequence subsets of linguistic information add a new level of complexity to the self-contained tutorials described above. A branching tutorial must account explicitly for the fact that students plunge into a given topic with widely different backgrounds and abilities. This kind of tutorial sets up a series of tasks that attempt to pin down the student's current competency in the topic area (e.g., reading a newspaper weather report, expressing wishes in the subjunctive) and adjust the sequence of the tutorial unit accordingly. The tutorial may even branch to remedial work from other tutorial segments. This approach can still be implemented with the help of an authoring language, although it requires a high degree of subject matter and instructional design expertise.

Generative Drill for Simple Structural Features

The term "generative CAI" refers to the use of computer logic to create

morphologically and syntactically correct linguistic forms as the basis for structural practice. At the simplest level, the machine can be made to conjugate the complete paradigms of verbs (to the extent that they have regular features). A functioning verb conjugator can then serve as the basis for a sentence generator. The sentence generator becomes the engine for structural exercises that have an almost infinite number of items generated at random on a given topic.

A project of this sort is probably not appropriate for an authoring language, but rather should be seen as an intermediate-level programming task. The more complex the linguistic features and the more extensive the lexicon used, the greater the challenge to the programmer, who will have to acquire a relatively thorough understanding of the structures of the target language. The computer that generates correct paradigms and sentences can be programmed to generate incorrect forms as well, in anticipation of student errors. The power of the generative program is therefore twofold: it can pour forth practice materials almost endlessly and it can invoke powerful diagnostic strategies to help the learner understand his or her weaknesses. Programs that attempt to do all this will quickly cross the boundary from development to research.

Simple Reading Comprehension Exercises

Very few reading packages are available that go beyond simple multiple-choice or fill-in-the-blank comprehension checking. Perhaps software developers have allowed their creative insights to be throttled by fears that their efforts might be termed "page turners," a label applied indiscriminately to any software that displays chunks of text on the screen. Advances in reading research already point to a wide range of skill-building activities that lend themselves to interactive computer delivery. Like generative CAI, reading programs can begin with the simpler forms of text display and move steadily into experimental areas that offer variant versions of the text; branch to easier and harder texts; supply lexical, structural, and cultural commentary; and much more. In contrast to generative CAI, reading projects probably lend themselves to implementation via a sophisticated authoring language.

Adventure Games Using Preprogrammed Models

Adventure games can form the vehicle for highly motivated learning sessions in reading comprehension and culture. The student becomes immersed in an environment vicariously, instead of focusing only on formal practice; vicarious cultural immersion is more likely to evoke the use of the language for a communicative purpose. Kossuth (6) presents a

thoughtful case for more systematic work with adventure formats, relating their use to Krashen's Input and the Affective Filter Hypotheses and describing in some detail the use of adventure authoring tools. Sanders and Sanders (14) discuss the prospects for intelligent foreign language games using the language-understanding techniques of artificial intelligence. (See also Chapter 8 in this volume, by Underwood.)

Programming adventure games and other simulations can be a demanding assignment. The simulated universe contains persons, objects, places, and actions that must be carefully tracked by the program. Communication between the learner and the computer can range from well-disguised multiple choice alternatives to machine "understanding" of hundreds or thousands of expressions. In most cases, representation of events in the simulated reality requires elaborate programming of graphics. Fortunately, adventure game packages are beginning to appear commercially that provide all of the graphics and the concealed juggling of simulation data. The instructor/author is permitted to create new scenarios, design new environments, and weave all new plots for the learner to unravel. Although there may be drawbacks to such adventure construction kits—lack of foreign characters, culturally inauthentic graphics—the advantages in terms of personalizing and contextualizing language practice probably outweigh the current disadvantages.

As the above series of typical development projects shows, the creation of original courseware can place major demands on the subject-matter specialist, i.e., the language teacher. The third level of project complexity, verging on the experimental, demands even greater time and resources and has an added element of risk, since there is seldom a guarantee of success.

Research Projects

Research projects are understood here as those large-scale or experimental projects that cannot be accomplished without a significant amount of original programming or whose success depends on the complex interaction of many modules or several media. Many of the possible projects described below will also require very careful evaluation to measure the effectiveness of the learning strategies involved.

Comprehensive Tutorials with Recordkeeping

One of the prime motivations for developing tutorial programs lies in the prospect of banning the explanatory aspects of language teaching from the classroom in favor of applying language skills in meaningful contexts.

In its extreme form, a full-scale tutorial could be used to deliver individualized instruction courses, correspondence courses and related forms of distance learning, and other nontraditional kinds of language study. The tutorial projects described above assume that computer work is conducted in parallel with a traditional course and that a human tutor consistently measures student progress. The final step in designing a full-fledged tutorial package—one that might take a learner from complete unfamiliarity with the target language to, say, the Intermediate–High level (1+) on the ACTFL/ETS scale requires careful tracking of student progress.

Recordkeeping may involve data collection via pretesting and posttesting to evaluate the learner's progress; it also requires a secure system to make a record of each step of the learner's work. The responsibilities of the instructional designer are greatly increased when the computer is asked both to instruct and to evaluate student progress. The tutorial interaction must be pedagogically correct and effective; the methods used to measure the outcome must also be reliable. Success in the tutorial must bear a fair relationship to the accomplishments demanded of students in traditional instructional settings. This implies extensive testing under controlled conditions over a long period of time. Such testing may show that specific modules of the tutorial are ineffective and should be redesigned and reprogrammed. The difficulties suggested here may explain why no completely self-contained tutorial course is currently available for any foreign language. Even when tutorials of this nature do emerge, they will doubtless raise questions about the specific proficiencies they claim to foster. In the end, a variety of tutorial software will be needed that employs different means to convey (1) structural knowledge, (2) reading proficiency, and (3) writing skills.

Drill or Tutorial with Animated Graphics

Text-based presentations seldom exploit the increasing capacity of powerful microcomputers to use color, shape, and movement to clarify relationships and illustrate content. Most authoring systems offer rudimentary tools for what can be called textual graphics—movement of letters, boxes, colors to establish links between concepts and words. Projects that go beyond text animation and static illustration require sophisticated programming techniques and are therefore extremely costly. Software for authoring seems to lag far behind the power of today's machines, but developers are well advised to keep track of illustration software (with names like "graphics magic kit," "computer paintbox," etc.) and ever more powerful authoring systems that include animation features.

Computer graphics in foreign language CAI may also engender some

controversy because primitive illustrations seldom capture the cultural authenticity found in visual materials in our textbooks. Commercial publishers may be able to overcome these limitations by commissioning animation from foreign artists. Language teachers can now approach the problem by using graphics-digitizing devices to actually transfer authentic images from print or video into their computer systems for use in structural practice, tutorials, or cultural simulations.

Creating Authoring Tools for Grammar and Reading

Most large-scale projects attempted today are designed around a fixed set of activities and interactions that appear throughout a whole series of instructional modules. The computer programs and routines that support these tasks form what is known as a "driver." The driver manipulates variable lesson content provided by the language teacher/author. One approach to creating CALL software for a large segment of the curriculum is to write original driver programs with the intent that they will be used as a template or authoring system.

Designing a new authoring system assumes that small-scale pilot work has already been done to discover an effective set of interaction types for a given purpose. For example, an arsenal of reading comprehension activities—including prereading tasks, skimming and scanning activities, reading utilities such as glossaries, and postreading comprehension checks—could be carefully prototyped for a single intermediate-level text in Spanish. An authoring language can be used to expedite this preliminary process. Once a satisfactory set of learning tasks has been worked out and tested by students, it is possible to accelerate the development process radically by generalizing the software so that the original text can be replaced by *any* Spanish text. In the case of reading comprehension programs, the resulting template will probably be useful for reading programs in other European languages.

By approaching the tasks of software development with the goal of creating a specially designed authoring system, one can reduce long-term costs and improve the reliability of the ultimate products. Whether one creates a new authoring system using a high-level authoring language or resorts to programming in a language like BASIC, Pascal, or C will depend largely on the power of the authoring system being considered and on the complexity of the tasks envisioned by the teacher/author.

Large Databases for Structure and Reading Practice

Once a specialized authoring system has been developed and tested thoroughly with a small set of materials, the natural next step is to exploit its capabilities to the fullest. This step should be considered a project in

itself, for it requires the involvement of other faculty, assistants with excellent linguistic and pedagogical training, native-speaker proofreaders, program testers, and a documentation editor. Work on this scale requires extra equipment and a project manager. Any authoring system with a high degree of flexibility should be considered a major resource for the campus and beyond, since using the authoring system will *always* be more cost-effective than designing new software from scratch. Granting agencies and commercial textbook publishers can be approached with strong arguments for funding to implement specific sets of material using the authoring system. The success of such proposals rests on the quality of the authoring system and on the institution's ability to carry a large project through to completion on schedule.

Complex Simulations of Target Culture Situations

Designing an authentic cultural simulation for the computer can be likened to producing a major motion picture or writing the Great American Novel. The content and language must carefully replicate the culture of a foreign country. Means must be found to present the situation to the learner via text, graphics, sound, or video. Unlike a movie or a novel, alternate scenarios must be plotted and brought to a conclusion. Fault can be found with many of the simulations attempted in the past because they ignore—knowingly or unknowingly—the imperative of authenticity in the interest of exploring the dynamics and technology of the simulation. Despite these caveats, simulations remain extremely attractive to the language teacher because even the weakest simulation attempts to give the student a controlled context for language usage. This moves CALL a large step beyond discrete-point structural practice and reflects the best insights of language-acquisition theory.

Language Understanding Projects

The projects outlined above have a fundamental limitation: they treat student input as data rather than as communication. The emergence of more powerful microcomputers with more advanced programming languages as C, Lisp, and Prolog has begun to justify research projects that attempt to analyze linguistic input in terms of its syntactic structures and ultimately its semantic content. Successful packages have already been published that accept a wide range of student expression, parse its structure, and even evaluate whether it is meaningful. In his discussion of artificial intelligence (Chapter 8 in this volume), Underwood outlines in detail the tasks and promise of linguistically based approaches to CALL.

Validated Placement and Proficiency Tests

The impetus toward developing testing instruments has become stronger with the advent of a wide consensus in the profession about the need for national standards in language proficiency. Mastery of structural features of a language and proficiency in reading skills are the two areas most likely to benefit from this trend through CALL. Listening comprehension tests will also emerge with the help of advanced audio and video technologies; speaking and writing tests, however, will feel little impact from technology.

The use of the computer in language testing, however, is still in its infancy, especially where it relates to established criteria such as the ACTFL/ETS *Guidelines*. The need for computer-based reading proficiency tests that attain the adaptive flexibility of the ACTFL Oral Proficiency Interview has been recognized, but full-fledged implementation lies in the future. Despite optimistic projections by Wyatt (16) and Pusack (10), serious questions can be raised about the reliability of adaptive tests that present different sequences of items to different students. It is clear that the programming difficulties inherent in such tests are small compared to the pedagogical concerns.

The list of potential areas for CALL research projects can be extended infinitely, especially into high-technology realms where new devices make possible ever more powerful types of student–computer interaction. The above examples, however, remain within the scope of proven hardware and software and thus deserve careful consideration by advanced users of CALL.

Conclusion

Providing CALL means engaging in a long-term process. As teacher-training programs begin to take technology into account, teachers will begin their careers with a foundation for building CALL into the curriculum. Everyone concerned, however, must explore the possibilities systematically and exploit intelligently what has been done in the past. Too many projects have simply replicated previous work. Hope, Taylor, and Pusack (5) include an extensive annotated bibliography to guide more extensive readings. Marty (7) draws a number of valuable conclusions about CALL from his extensive experience during the development of high-quality French lessons on the PLATO system. Wyatt (17) offers a perspective from the ESL curriculum that has many areas of overlap with foreign-language computing. Impetus from the wider educational context can also help in the effective design and use of instructional computing. Two highly recommended handbooks with a larger perspective are Alessi and Trollip (1) and O'Shea and Self (8).

Ultimately, progress in CALL rests with only one group: language teachers themselves. Ideas from professional design experts, algorithms from state-of-the-art programmers, and market-oriented products from commercial publishers will all miss the mark if teachers do not articulate their needs. Whether it be by critiquing packages, implementing existing CALL materials, authoring new programs, or engaging in long-term experiments, all foreign language teachers can play a decisive role by insisting that they know good software when they see it. Good CALL software has one essential property: it incorporates our best knowledge about how today's learners learn languages.

References, Problems and Prospects in Foreign Language Computing

1. Alessi, Stephen M., and Stanley R. Trollip. *Computer-Based Instruction: Methods and Development.* Englewood Cliffs, NJ: Prentice-Hall, 1985.
2. Bigham, Diane, Gene Portwood, and Lauren Elliott. *Where in the World Is Carmen Sandiego?* San Rafael, CA: Broderbund Software, 1985.
3. Cornick, Lisa B. "Evaluating Foreign-Language CAI: A Checklist." *Die Unterrichtspraxis* 17 (1984):315–18.
4. Hamerstrom, Helen, Gladys Lipson, and Sherry Suter. "Computers in the Foreign Language Classroom: No Longer a Question." *CALICO Journal* 3, 1 (1985):19–21, 48.
5. Hope, Geoffrey R., Henry F. Taylor, and James P. Pusack. *Using Computers in Teaching Foreign Languages.* Language and Education: Theory and Practice, no. 57. Orlando, FL: Harcourt Brace Jovanovich, 1984.
6. Kounuth, Karen C. "Using the Adventure Formats for CALL." *CALICO Journal* 3, 2 (1985):13–17.
7. Marty, Fernand. "Reflections on the Use of Computers in Second-Language Acquisition." *System* 9, 2 (1981):85–98; 10, 1 (1982):1–11.
8. O'Shea, Tim, and John Self. *Learning and Teaching with Computers: Artificial Intelligence in Education.* Englewood Cliff, NJ: Prentice-Hall, 1983.
9. Otto, Sue E. K. and James P. Pusack. "Tools for Creating Foreign Language CAI," pp. 96–108 in Patricia B. Westphal, ed., *Meeting the Call for Excellence in the Foreign Language Classroom.* Proceedings of the Central States Conference on the Teaching of Foreign Languages. Lincolnwood, IL: National Textbook Company, 1985.
10. Pusack, James P. "The Interactive Computer Testing of Reading Proficiency." *Foreign Language Annals* 17, 4 (1984):415–19.
11. _____. "Answer-Processing and Error Correction in Foreign Language CAI." *System* 11, 1 (1983):53–64.
12. _____. *DASHER: An Answer Processor for Language Study.* Oakdale, IA: CONDUIT, 1983.
13. Pusack, James P., and Sue E. K. Otto. "Blueprint for a Comprehensive Foreign Language CAI Curriculum." *Computers and the Humanities* 18 (1984):195–204.
14. Sanders, Alton F., and Ruth H. Sanders. "Spion: 'Intelligent' Games for German Language Teaching," pp. 141–46 in *Foreign Language Instructional Technology Conference Proceedings, 21–24 September, 1982.* Monterey, CA: Defense Language Institute, 1983. [EDRS: ED 236 910.]
15. Tenczar, Paul et al. *TenCORE Language Authoring System.* Champaign, IL: Computer Teaching Corporation, 1984.

16. Wyatt, David H. "Computer-Assisted Teaching and Testing of Reading and Listening." *Foreign Language Annals* 17, 4 (1984):393–407.
17. _____. *Computers and ESL.* Language and Education: Theory and Practice, no. 56. New York: Harcourt Brace Jovanovich, 1984.